

MINIMIZING INTERACTION COSTS AMONG COMPONENTS OF COMPUTER PROGRAMS

Cross-Reference To Related Applications

This application claims the benefit of United States Provisional Application Number 60/267,573, filed February 9, 2001.

The present application is related to the present inventors' applications entitled "Characterization Of Objects Of A Computer Program While Running Same" Serial No. _____ (IBM Docket YOR920020050), and "Program Components Having Multiple Selectable Implementations" Serial No. _____ (IBM Docket YOR920020023), which were filed on the same day as the present application. These related application are incorporated herein by reference.

Field of the Invention

This invention generally relates to the field of optimization of computer programs, and more particularly relates to a computer system that transforms programs so that they run more quickly or otherwise at lower cost, but produce the same results, by minimizing costs of interaction among program components.

Description of Related Art

Programmers are increasingly programming by component assembly. This is despite the fact that the use of components tend to decrease the performance of the program. The programmers who design the components do not know

anything about the environment the components will be used in, and can therefore not tailor the components for each specific use. The efficiency of a program suffers from the use of such generic components, since it fundamentally depends on the interactions between the components. The performance of a component based program is therefore often less than optimal. Optimizing component interaction is a fundamental problem in constructing efficient component-oriented programs.

Many computer programs , which consist of a number of program components, manipulate implementation properties such as string representations and data structure for which any of a number of implementation properties can be used. For example, string representations that can be used include: UNICODE, ASCII, and EBCDIC. As another example, data structures that can be used include: trees, compressed files and hash tables. In fact, in many database and virtual machine benchmarks, substantial amounts of time are lost in converting data representation values back and forth between a number of different representations.

One problem solved by this invention is that of minimizing the number of transformations of implementation property values, such as data structure values, back and forth over the course of a run of a computer program.

For consistency of definition in the context of the present application, it should be understood that the term "property", with respect to an object or component of a computer program, is broad, and includes narrower terms such as "location", "parameter", and "implementation ". In turn, " implementation" includes "data representation" such as "string representation" (e.g. ASCII, EBCDIC, UNICODE) and "data structure" (e.g. hash, tree, compressed). Thus, it will be understood

that "implementation" does not encompass "location", nor "parameter" within its meaning. Moreover, in the context of the present invention, "object", "entity", and "component" shall be interpreted as having substantially the same meaning, while "library" shall be understood to mean a group of object definitions or component definitions.

SUMMARY OF THE INVENTION

The present invention broadly provides a method for minimizing total cost of interaction among components of a computer program, each of the components being characterized by at least one implementation property. The method comprises the steps of:

- a) carrying out at least a partial run of the program;
- b) monitoring the at least partial run of the program to measure an amount of interaction between each pair of components;
- c) determining a cost of interaction between each pair of interacting components;
- d) determining a choice of implementation properties which minimizes total cost of the at least partial run; and
- e) assigning that choice of implementation properties to the components for a subsequent at least partial run of the program.

According to a preferred embodiment, the implementation property comprises a choice of string representation (e.g. ASCII, UNICODE, EBCDIC) of a component, the amount of interaction measured in step (b) comprising a frequency of interaction between each pair of interacting components. The aforesaid cost of interaction may comprise a function of the aforesaid frequency

and a cost of converting any differing string representations of the pair to a common string representation. .

5 According to another preferred embodiment, the aforesaid implementation property comprises a choice of data structure (e.g. hash tree, and compressed data structures) of a component, the aforesaid amount of interaction measured in step (b) comprising a frequency of interaction between each pair of interacting components; the aforesaid cost of interaction comprising a function of the aforesaid frequency and a cost of converting any differing choices of data
10 structures of the pair to a common choice of data structure.

15 Preferably, step (d) of determining the choice is carried out by a mathematical solution, as by building a graph with nodes representing program components and edges that join adjacent nodes representing interaction therebetween, each edge being characterized by a cost of each interaction, then using a graph cutting technique to find a minimum cut of the graph. Such graph cutting techniques are described in three United States patent applications Numbers 09/676,423 by Rajan et al, 09/676,424 by Wegman et al, and 09/676,425 by
20 Roth et al, all filed on September 29, 2000.

25 The invention also comprises a computer readable medium including computer instructions for carrying out each method disclosed herein for minimizing total cost of interaction among components of a computer program running on a computer system.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic illustration, sometimes called an OAG (Object Affinity Graph), of program components, implementation properties, and interactions among components, according to an exemplary preferred embodiment of the present invention.

FIG. 2 is a schematic illustration derived from FIG 1, showing a graph partition or graph cut that may minimize component interaction costs for a subsequent, at least partial, run of the program, according to the preferred embodiment of FIG 1.

FIG. 3 is a is a schematic illustration, sometimes called an OAG (Object Affinity Graph), of program components, implementation properties, and different interactions among components, according to another exemplary preferred embodiment of the present invention.

FIG. 4 is a schematic illustration derived from FIG 3, showing a graph partition or graph cut that may minimize component interaction costs for a subsequent, at least partial, run of the program, according to the preferred embodiment of FIG 3.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

According to a preferred embodiment of the present invention, a computer system and method (e.g. by instrumenting the program itself, or by instrumenting the system on which the program runs), monitors a representative run of a program (i.e. a run which is similar to the kind of run for which the program is to be optimized -- both in terms of the way the program is invoked, and in terms of

the kind of input data it is given). During the monitored run, the system notes the source variable, the destination variable, and the amount of data moved, for each operation in which data flows from one variable into another.

5 During or after the run, use this information to construct a graph as follows. For each variable seen during the run, create a non-terminal node corresponding to it. For each flow of data between two variables, create an edge between the two non-terminal nodes corresponding to the two variables, if such an edge does not already exist, and add to the weight of that edge an amount proportional to the
10 amount of data that flowed. For all possible representations of data values seen during the run, create a terminal node corresponding to each such representation. For each use of a variable in an interface to an external program that required that variable to have a particular implementation, one may add an edge between the non-terminal node corresponding to that variable and the
15 terminal node corresponding to the representation, with an infinite weight, if such an edge does not already exist.

Having constructed the graph, one may perform a multi-terminal minimum cut of the graph by any practical means. For each resulting partition, one per terminal
20 node, one should transform the program so that in future runs of the program, each variable corresponding to non-terminal node in the partition will use the representation corresponding to the terminal node of the partition.

25 EXAMPLE:

The following is a specific exemplary embodiment of the present invention
Consider the case in which we are trying to minimizing the cost of running a

program comprising program components a, b, c, d, e, and f, which may be implemented in two implementations, in this case the string representations ASCII and UNICODE, as shown in FIG 1. In this example, components a and b are in ASCII code, and components c and d are in UNICODE. The property we are seeking to determine is the data representations of component e and f. The first step is to run this program. For convenience we may run this program on one computer and monitor the interaction between components using a tool such as "Jinsight". This monitoring would provide the frequency of interaction between each pair of interacting components. This data, together with the cost based on transforming a component to ASCII and UNICODE, helps determine the total cost of interaction among components of the program. This cost of interaction is used to build the OAG (Object Affinity Graph) graph shown in FIG 2. The nodes in the graph of FIG 1 and FIG 2 are the components a, b, c, d, e, and f and the implementations ASCII and UNICODE. The edges of FIG 2 represent the frequency of interaction and the weights on the edges represent the cost of not implementing adjacent components in the same representation. These costs are a function of the frequency of interaction and the cost of each interaction.

In the example of FIG 1, the frequencies of interaction, based on monitoring of the running of the program, are 100 for component a with component e, 20 for component e with component d, 25 for component e with component c, and so on, as shown. The cost of each interaction may be 6 for not transforming component e to ASCII via component a and 6 for UNICODE via component d. The cost values for not transforming component f to ASCII via component b is 7, and so on, as shown in FIG 1.

The cost values shown in FIG 2 may then be derived from the frequency and weight values of FIG 1. For example, the cost between components a and e is 6

x 100 or 600, while the cost between e and c is 6 x 25 or 150. The cost of interaction may then be minimized in accordance with the graph cutting techniques described in the aforementioned three United States patent applications Numbers 09/676,423 by Rajan et al, 09/676,424 by Wegman et al, and 09/676,425 by Roth et al, all filed on September 29, 2000.

As will be understood, the OAG graph of FIG 2 should be cut on the side of component e away from component a because the cost of not implementing e in ASCII is 600, while the cost of not implementing e in UNICODE is only 270 = (120 + 150). Similarly, the cut should implement component f in UNICODE because, non-implementation in UNICODE costs 260, while the cost of not implementing f in ASCII is only 105, as shown in FIG 2.

FIG 3 and FIG 4 show how the cost-minimizing cut changes when the interaction costs change. Thus, when frequency of interaction between a and e changes from 100 to 40 in FIG 3, the overall cost drops in FIG 4 to 240 which is less than the UNICODE affinity, Thus component e should be implemented in UNICODE based on the dotted cut line of FIG 4, while component f should now be implemented in ASCII.

The future runs of the same program on this network of computers should be run with this implementation of components e and f to minimize the cost of running the program on the network.

As will now be understood from this example, the methods described in the three above-cited patent applications have the advantage of being able to find a cut quickly. This allows problems of a given size to be solved more quickly, and it allows larger problems to be solved within a given time constraint (problems which might not have been practical to even attempt to solve otherwise).

A graph consists of nodes and edges. Each edge connects two nodes. A weight is associated with each edge. Some of the nodes are designated as being "terminal" nodes. The others are designated as being "non-terminal" nodes.

5

For use in optimizing data structure value representation conversion: each terminal node can represent a possible representation; each non-terminal node can represent a variable (the representation used by a particular variable is fixed over the course of the entire run); and the weight of each edge between two non-terminal nodes can represent the amount of data that was assigned back and forth over the course of a run between the variables corresponding to the two nodes that the edge connects, or the amount of work needed to convert that data. An edge between a non-terminal node and a terminal node can represent the fact that, due to external interface requirements, the variable corresponding to the non-terminal node must use the representation corresponding to the terminal node. In this case, the edge would be given infinite weight.

10

15

A multiway cut of a graph is the set of edges which, when removed from the graph, results in there no longer being a path through the graph from any terminal to any other terminal. A multiway minimum cut of a graph is a cut, the sum of the weights of whose edges is less than or equal to that of any other such cut. Such a cut partitions a graph into one partition for each terminal node. The partition for a terminal node consist of all non-terminal nodes connected to it directly, or indirectly via other non-terminal nodes.

20

25

In optimizing data structure value representation conversion, the non-terminal nodes in a partition represent variables that should use the representation corresponding to the terminal node of the partition, in order to keep

representation conversions to a minimum, and hence to achieve best overall performance of the program.

5 The preferred embodiments of the present invention can be realized in hardware, software, or a combination of hardware and software. Any kind of computer system - or other apparatus adapted for carrying out the methods described herein - is suited. A typical combination of hardware and software could be a general purpose computer system with a computer program that, when being loaded and executed, controls the computer system such that it carries out the methods described herein.

10 The present invention can also be embedded in a computer program product, which comprises all the features enabling the implementation of the methods described herein, and which - when loaded in a computer system - is able to carry out these methods. Computer program means or computer program in the present context mean any expression, in any language, code or notation, of a set of instructions intended to cause a system having an information processing capability to perform a particular function either directly or after either or both of the following a) conversion to another language, code or, notation; and b) reproduction in a different material form.

20 Each computer system may include, inter alia, one or more computers and at least a computer readable medium allowing a computer to read data, instructions, messages or message packets, and other computer readable information from the computer readable medium. The computer readable medium may include nonvolatile memory, such as ROM, Flash memory, Disk drive memory, CD-ROM, and other permanent storage. Additionally, a computer medium may include, for example, volatile storage such as RAM, buffers, cache

memory, and network circuits. Furthermore, the computer readable medium may include computer readable information in a transitory state medium such as a network link and/or a network interface, including a wired network or a wireless network, that allow a computer to read such computer readable information.

5

Although specific embodiments of the invention have been disclosed, those having ordinary skill in the art will understand that changes can be made to the specific embodiments without departing from the spirit and scope of the invention. The scope of the invention is not to be restricted, therefore, to the specific embodiments, and it is intended that the appended claims cover any and all such applications, modifications, and embodiments within the scope of the present invention.

10

15

20

25